ELSEVIER

Theoretical Computer Science 181 (1997) 195–225

Theoretical Computer Science

A temporal logic for real-time partial ordering with named transactions

Farn Wang*

Institute of Information Science, Academia Sinica, Taipei, Taiwan 115, ROC

Abstract

We extend Lamport's partial-ordering models (Lamport, 1978) for real-time computing and invent to use the concept of named transactions to reference groups of related events. We then propose *transaction partial ordering logic* (*TPOL*) as a new specification language with special syntax and high-level semantics tailored to describe the interaction among transactions in a distributed real-time system. Finally, we examine TPOL satisfiability problems with different restrictions.

1. Introduction

This paper investigates special needs in modern computer system design and extends the expressive first-order logic (FOL) with proper syntax and semantics to design a specification language with built-in high-level structures in the hope of alleviating the burden of formal specification and benefiting from the technology of computer-aided verification.

Advanced computer systems may involve technologies like distributed computation, timing interactions, transaction processing, etc. As human ambition for even more powerful computer system construction never ceases, the tasks of specification and verification have grown rapidly and the delivery of reliable and safe systems is now at great risk. Challenged and fascinated by this issue, engineers and theoreticians in every aspects of the system sciences are now proposing all sorts of solutions. From the engineering side, people have developed various disciplines including *software engineering* [17] and *formal methods* [11, 20] which recommend that users specify interesting system behaviors in some guideline and then leave users on their own when it comes to system verification. From the viewpoint of theory, various frameworks have been adapted directly from classic mathematical theories, from *modal logics*, to *first-order logics* (*FOL*), to *higher-order logics*. On the positive side, they provide elegant

* Tel.: +886-2-7883799 ext. 2420; Fax +886-2-7824814; e-mail: farn@iis.sinica.edu.tw.
Supported by NSC, Taiwan, ROC under grant NSC 84-2213-E-001-012.

PII S0304-3975(96)00270-8

models and rigorous insight into the basic nature of the relevant problems, e.g. verification complexities, while on the negative side, they may not effectively use the characteristics of modern computer systems to alleviate the burden of formal specification and verification. As a result, some of the theories lack descriptive power for common behaviors while some easily incur in undecidable verification tasks.

C.B. Jones noted that [12]

> "It has long been realized that restrictions on programming languages are key to the development of concurrent programs".

It is our belief that by designing a specification language with higher-level syntax constructs and higher-level abstract semantics for *partial ordering* computing, we can get the following benefits in applying the research results of temporal logics to system design.

- *Necessary expressiveness for partial ordering.* The models of partial ordering are not only convenient for describing incomplete knowledge of temporal precedence, but also necessary for describing *message flows* and *causality* in distributed systems.
- *A solid theory for specification and verification of modern computer systems.* Only with such a theory can engineers benefit from the technology of formal specification and automated verification.
- *Better representation of higher-level human reasoning in system design.* Binary low-level representations of some system behaviors can be bulky and may hide the simplicity of system design ideas.
- *More built-in semantics and less verification overhead.* Bulky low-level representations can create much computation overhead in verification.

It is the purpose of this paper to design a new language, called *transaction partial ordering logic* (*TPOL*), for the specification and verification of distributed real-time systems with partial ordering behavior. Our contributions include the following:

- We naturally extend Lamport's partial ordering models of distributed systems [13] with real-time events. Then asynchronous inequalities [22] are naturally modified as a notation for real-time communications in our new partial ordering computing models.
- We investigate use of the concept of named transactions and allow specification of complicated interactions among events from different named transactions. This concept of giving names to transactions and events is compatible with the way people describe the behaviors of complicated systems.

As we have noted, partial ordering is important in describing message flows and causality in distributed systems. By giving names to different transactions and events, TPOL naturally facilitates specification of sources and destinations of such message flows in real-time partial ordering computing.

This design consideration gives TPOL not only additional expressiveness compared with some previous works, but also a background theory to support computer-aided verification. In our approach, the user first describes the system behavior in a TPOL formula, say S, and then specifies the system requirement in another TPOL formula,

say P. To check if system S does satisfy requirement P, we simply compute the satisfiability of $S \wedge \neg P$. If it is satisfiable, this means that P can be violated for some partial ordering schedules of system S, and that the system design is not safe. One way to do this is to use the theorem provers for first-order logics. In Sections 4 and 6, we will also discuss the possibility of decidable verification for subclasses of TPOL. Note that since TPOL allows named transactions in a partial ordering environment, both P and S can be very high-level and succinct. One goal of this research is to utilize this advantage to significantly reduce verification overhead. Also note that in our approach, there is really not much distinction between S and P. The satisfiability of $S \wedge \neg P$ is equivalent to that of $(S \wedge \neg P) \wedge \neg \mathit{false}$.

In the following, we will give the user a little taste of TPOL formulas. The formal definition will follow in Section 2. More intuition behind the TPOL syntax and semantics will be explained in Section 3.

Example 1. Consider a computer company which has two departments, manufacturing and delivery. Each department has its own clock. The customers place their orders according to the customers' clock. Each transaction has three events. An *Order* event happens when a customer places an order by phone. A *Make* event happens when a computer is manufactured. A *Delivery* event happens when a computer is delivered. The company wants to enforce the policy that "every order will be serviced within the deadline of five time units according to the delivery service's clock." This can be stated in TPOL as

$$\bigtriangledown\{[x]\}(\mathrm{Make}[x] \prec \mathrm{Delivery}[x] \wedge \mathrm{Order}[x] \prec \mathrm{Delivery}[x] \wedge \mathrm{Delivery}[x] - 5 \preccurlyeq \mathrm{Order}[x])$$

Here $\bigtriangledown$ is a modal operator adopted from [23] that universally quantifies over a set of transactions together with their names. x is a variable for transaction names. Delivery$[x]$, Order$[x]$, and Make$[x]$ are the three events of transaction $[x]$. Delivery$[x] - 5$ is the event that happens within five time units, according to the clock at which Delivery$[x]$ is observed, before event Delivery$[x]$. $\prec$ and $\preccurlyeq$ are partial ordering relations and mean "earlier than" and "no later than," respectively. We use $\prec$ to mimic the "precede" ($\rightarrow$) relation in Lamport's partial ordering models [13], which can be interpreted as a notation for communications which can take any positive amount of time to complete. On the other hand, $\preccurlyeq$ can be interpreted as a notation for communications which can take any "nonnegative" amount of time to complete.

There are several nice properties of this formula. Firstly, it does not specify the temporal precedence between Make$[x]$ and Order$[x]$, and this means that the manufacturer is allowed to produce a computer before an order is placed for it to take advantage of the manufacturing capacity. Secondly, since we use name variable x in the formula, there is no confusion about which *Make* is for which *Order*. Thirdly, the manufacturing department and delivery department have the scheduling freedom to decide which *Order* will be serviced first. *Order*s do not have to be serviced on a first-come-first-served or first-come-last-served basis, nor are there any other restrictions. The two departments can use this freedom to maximize their productivity.

The new modal operator $\triangledown$ is adopted here not only for syntax conciseness but also for distributed computation intuitiveness. Here we give two examples to argue for it.

Example 2 (*The conciseness of the modal operator*). The new modal operator imposes temporal precedence among the quantified events when nested quantification is used. For example, we may want to say that "if we have the experience that an item is produced before it is ordered, then the items are hot and we should deliver them quickly after their production (actually in 5 time units)":

$$\triangledown\{[x]\}(\text{Make}[x] \prec \text{Order}[x] \rightarrow \triangledown\{[y]\}\text{Delivery}[y] \leqslant \text{Make}[y] + 5).$$

By intuition, such a "timely delivery" requirement should be enforced only after an experience of a "hot item" is sensed. Thus it is natural to ask that event Make[y] happen no earlier than event Make[x]. If we instead use the first-order logic quantification $\forall$ here, the formula will become much clumsier and blur the high-level behavior structure.

Example 3 (*The distributed computation intuitiveness*). The new modal operator treats the quantified events as being within the same scope as set elements. For example, we may want to say, "Order events and delivery events should not happen too close to each other (3 time units apart according to the delivery service's clock)":

$$\triangledown\{[x],[y]\}(\text{Delivery}[x] - 3 \leqslant \text{Order}[y] \leqslant \text{Delivery}[x] + 3)$$

Note that since the two event types are recorded according to two different clocks, there may be no precedence relation between an *Order* event and *Delivery* event. Thus, it is not possible to write the same specification with some linear-order semantics like the state sequence of traditional temporal logics. Thus, we believe that such a behavior can best be described by quantification on sets of events instead on states in linear sequences. Our modal operator helps to clarify this intuition.

In the following, we shall use several subsections to describe the common behaviors of modern computer systems, previous work, and the organization of the rest of this paper.

1.1. What is happening in modern computer systems?

Modern computer systems may consist of many processors geographically close or far-away from each other. Each processor is called a *site* in the distributed processing and may have its own *local clocks*. Synchronization among the sites can be tremendously difficult due to incomplete knowledge of global states, unexpected message delay, different time metrics, and unstable jitter in clock pulses. The fulfillment of any task in the systems requires timely consortium of various activities in several processors.

Furthermore, in a distributed environment, there is no global clock reference to all the sites. Each site has to rely on its local clock to observe the ordering among

events. Thus, partial ordering among the events is not only natural, but also intrinsic to the formal specification. Beside its expressiveness for temporal precedence, partial ordering is both convenient and mandatory when describing *message flows* and *causality*.

One useful concept developed to enhance the reliability of such system designs is *transaction processing*, which is nowadays almost completely accepted in the theory and practice of distributed system design. A *transaction* is a group of events such that outside the group, the transaction is treated atomically. The concept of a transaction has been proven very useful in simplifying design considerations by insulating the interaction among internal events from interference due to peer transactions.

1.2. Related work

With the increasing complexity of hardware and increasing demand for artificial intelligence capability [22], the benefit of using programming language design to aid computer-aided verification is becoming more and more evident.

The design of traditional programming languages does not emphasize the necessity of system verification and validation. The safety and reliability of systems are usually checked by using external methods, like those developed in software engineering [17]. As a result, the formal semantics of system specification and programming languages usually seems ad hoc, and complete system verification becomes almost impossible.

In recent years, the technique and theory of temporal logic [1, 3, 4, 8, 9, 15, 22, 24, 23] has attracted attention from academia and industry because of its potential for assisting with specification and verification of system designs. The prevalent approach in this research area is that of linear-time and branching-time propositional temporal logics and their variations [1, 3, 4, 15, 22]. Propositional temporal logics are excellent and elegant languages which remove the barrier to understanding the theoretical aspects of language design and computer-aided verification. However, they use state sequences, which is the concept of total ordering as models, and do not provide high-level ability to describe message flows and causality in distributed systems. Thus, inevitably, this approach is usually shackled by insufficient expressiveness and succinctness in specification and low efficiency in verification due to bulky low-level specifications which easily hide the high-level behavior structures. For example, some common system behaviors, like those of stacks and queues, cannot be expressed in propositional temporal logics.

Process algebra is also based on global sequences and shares most of the drawbacks of propositional temporal logics. However, at users' discretion, verification efficiency can be enhanced by using the hiding and composition operators.

Another approach in real-time temporal logic is that of event-based temporal logic. Typical work includes RTL [8, 9, 24] and AREL [23]. In systems described by such logic, there are many event types and many clocks. They use sequential *occurrence indices*, which are basically positive integers, to name different events of a type. Given

a schedule and an event type e, the ath event of type e counted from the beginning of the schedule is given occurrence index a and is denoted as $e(a)$. In RTL and AREL, we may use quantified variables to reference the occurrence indices of events. Such semantics make them a good choice for describing certain system behaviors, including those of queues, and also give them expressiveness that is not known to be achievable in traditional propositional temporal logic. Still the behaviors of stacks is not known to be expressible in RTL and AREL.

First-order logics (FOL) and first-order arithmetics (FOA) are also alternatives in specifying and verifying computer systems [2, 5, 18]. The advantage is the huge theory arsenal backing up the approach. The disadvantage is that very little characteristic knowledge of computer systems is built in. As a result, specifications in this approach can be bulky, and almost all nontrivial verification tasks expressed in the framework are of undecidable complexities. Compared to FOL, the contribution of this work is twofold. First, as we have argued in Examples 2 and 3, TPOL better identifies the behavior structure of distributed computing than does FOL. Second, as will be shown in Section 6, we are also able to identify a proper decidable subclass of TPOL for verification of distributed system behaviors.

From the engineering side, people have developed formal methods including VDM [11], Z notations [20], etc. With support from many companies, various successful applications have been reported. However, there still is not a general theory for automated verification in this approach. Engineers adopting formal methods are left on their own for verification of their system designs.

1.3. Organization of the rest of this paper

Section 2 introduces the syntax and semantics of TPOL formulas. Section 3 gives physical meanings to the computer behaviors (i.e. models) described by TPOL formulas. Sections 4–6 discuss the complexities of TPOL satisfiability problems with different restrictions. We shall show that the problem is EXPSPACE-hard when only one event type is allowed and is undecidable when only two event types are allowed. This second proof somewhat extends the frontier of our understanding of the complexity structure of event-based temporal logic decision problems. (A previous proof in [23] for event-based logic AREL uses an unbounded number of event types.) This restriction also implies that by simply restricting the number of event types used, no nontrivial languages may be obtained. This observation leads to the design of another restriction, the *synchronized bounded span* (*SBS*) restriction, described in Section 6. With the SBS restriction, we will show that the satisfiability problem of TPOL becomes EXPSPACE-complete. Special care has been taken in designing the presentation of the SBS restriction so that during verification, it can be enforced in tableau construction by simple checking routines on the tableau nodes and that it should incur minimum overhead during verification. Section 7 contains our conclusions.

2. Definition of transaction partial ordering logic

Section 2.1 gives the syntax of TPOL specifications. Section 2.2 defines its formal semantics. In the next section, we shall give the intuition for the design of its formal semantics.

2.1. Syntax of TPOL

A TPOL specification S consists of two parts. The first part specifies the number of clocks, the set of event types, and to which clock each event type is specific. Given a system described by a TPOL formula S, we assume that E_S is the set of event types, and that H_S is the number of clocks. Given a system of H_S clocks, we shall use integers $1,\ldots,H_S$ to index the clocks. We shall also assume that mapping $\iota: E_S \mapsto \{1,\ldots,H_S\}$ is given such that for all $e \in E_S$, $\iota(e)$ is the index of the clock to which events of type e are all specific.

The second part is *the temporal behavior description*, given by a well-formed TPOL formula. Unless there is ambiguity, we shall omit the first part from now on in the following discussion.

The syntax of a well-formed TPOL formula S is given below. e, f are event type names, $x, y, x_1, \ldots, x_m$ are *name variables*, m is a nonnegative integer constant, c, d are integer constants, and $\triangle$ is the new modal operator for transactions in partial ordering schedules:

$$S ::= \mathit{true} \mid e[x] + c \leqslant f[y] + d \mid \neg S_1 \mid S_1 \vee S_2 \mid \triangle\{[x_1],\ldots,[x_m]\}S_1$$

In $\triangle\{[x_1],\ldots,[x_m]\}S_1$, we use $\{[x_1],\ldots,[x_m]\}$ to quantify a *set* of transactions. Thus, it is legal to write the same formula as $\triangle\{[x_i] \mid 1 \leqslant i \leqslant m\}S_1$. We have the notions of *free* and *bound* variables and *scopes* as in first-order logic. In $\triangle\{[x_1],\ldots,[x_m]\}S_1$, $\{[x_1],\ldots,[x_m]\}$ serves to quantify $x_1,\ldots,x_m$ over the scope S_1. All occurrences of $x_1,\ldots,x_m$ in $\{[x_1],\ldots,[x_m]\}S_1$ are bound. An unbound variable in a TPOL formula is a free variable.

We shall also use *false*, $\triangledown\{[x_1],\ldots,[x_m]\}S_1$, $S_1 \wedge S_2$, $S_1 \rightarrow S_2$ as shorthand for $\neg \mathit{true}, \neg\triangle\{[x_1],\ldots,[x_m]\}\neg S_1, \neg(\neg S_1 \vee \neg S_2), (\neg S_1) \vee S_2$, respectively. For convenience, for any two terms A, B, we shall also use $A \prec B, A \succcurlyeq B, A \succ B$, and $A \asymp B$ as shorthand for $A \leqslant B \wedge \neg(B \leqslant A), B \leqslant A, B \prec A$, and $A \leqslant B \wedge A \geqslant B$, respectively. Parentheses will be used whenever it is necessary to disambiguate the syntax.

2.2. Semantics of TPOL

The models of TPOL formulas are called *partial ordering schedules* and consist of two types of basic atoms, *names* and *local clock readings*. A name is a character string used to distinguish activities of the same type in a computation and associate related events in the same transaction. A local clock reading is a temporal mark that denotes the instant at which a reading of a local clock is read. Given $1 \leqslant i \leqslant H_S$, we use a_i to denote the reading value a of clock i. There is a special local clock reading $\perp$

which means *undefined*. For every $a \in \mathcal{N}, 1 \leqslant i \leqslant H_S$, we let $a_i + \perp = \perp$ and $\perp + a_i = \perp$. However, for every integer a, b, and $1 \leqslant i \leqslant H_S$, we let $a_i + b_i = (a+b)_i$ and $a_i + c = (a+c)_i$.

TPOL formulas are interpreted over *partial ordering schedules* defined in the following way.

Definition 1 (*Partial ordering schedules*). Given a TPOL formula S, $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ is a partial ordering schedule for S iff

- $\Omega \subseteq \{a_i \mid a \in \mathcal{N}; 1 \leqslant i \leqslant H_S\}$ is a set of local clock readings.
- $\Gamma \subseteq \{(a_i, b_j) \mid a_i, b_j \in \Omega\}$ defines the partial ordering relation and satisfies the following conditions.
  - (*Integer ordering*) For all $a_i, b_i \in \Omega$, $a \leqslant b \leftrightarrow (a_i, b_i) \in \Gamma$.
  - (*Transitivity*) For all $(a_i, b_j), (b_j, c_k) \in \Gamma$, $(a_i, c_k) \in \Gamma$.
  - (*Progressiveness*) For all $a_i \in \Omega$, $1 \leqslant j \leqslant H_S$, there is a $b_j \in \Theta$ such that $(a_i, b_j) \in \Gamma$.

  From now on, for every $(a_i, b_j) \in \Gamma$, we shall write $\Gamma \models a_i \leqslant b_j$ and $\Psi \models a_i \leqslant b_j$. Note especially that for all $a_i \in \Gamma$, $\Gamma \not\models a_i \leqslant \perp$, $\Gamma \not\models \perp \leqslant a_i$, and $\Gamma \not\models \perp \leqslant \perp$.
- Θ is a set of names.
- Λ is a mapping from $E_S \times \Theta$ to $\{\perp\} \cup \Omega$ such that for all $e \in E_S$ and $p \in \Theta$, $(\Lambda(e, p) \neq \perp \rightarrow \exists a \in \mathcal{N}(\Lambda(e, p) = a_{\iota(e)}))$.

Given $e, f \in E_S$, $p, q \in \Theta$, $\Lambda(e, p) = a_i$, $\Lambda(f, q) = b_j$, and $\Gamma \models (a+c)_i \leqslant (b+d)_j$, we may also write $\Gamma \models e[p] + c \leqslant f[q] + d$ and $\Psi \models e[p] + c \leqslant f[q] + d$.

Also, since $\Gamma \not\models \perp \leqslant \perp$, we shall use $e[p] \neq \perp$ as shorthand for $e[p] \leqslant e[p]$, which tests the existence of $e[p]$ in a given partial ordering schedule.

We introduce the following notations to formally define the semantics of TPOL formulas. Suppose we are given a partial ordering schedule $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ for a TPOL formula S and a name $p \in \Theta$. *Transaction* p, denoted as $[p]$, is defined as the set $\{e[p] \mid e \in E_S\}$ of terms. For each transaction $[p]$, we can assign a *value*, denoted as $\Lambda[p]$, to it in Ψ, i.e. $\Lambda[p] = \{\Lambda(e, p) \mid e \in E_S; \Lambda(e, p) \neq \perp\}$. $\Lambda[p]$ is called the value of transaction $[p]$ in Ψ and describes the set of event times of $[p]$ in Ψ.

Suppose we are given a partial ordering schedule $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ and a subset $\Omega' \subseteq \Omega$. start(Ψ, Ω') is the set of the earliest events in Ω' according to partial ordering Γ, i.e.

$$\text{start}(\Psi, \Omega') = \{v \mid v \in \Omega';\ v \neq \perp;\ \forall v' \in \Omega'(\Gamma \not\models v \succ v')\}$$

Given a set L of local clock reading sets, we define start$(\Psi, L) = \bigcup_{\Omega' \in L} \text{start}(\Psi, \Omega')$. Also, $\mathcal{E}[x_i \leftarrow p_i \mid 1 \leqslant i \leqslant m]$ denotes the environment that agrees with $\mathcal{E}$ on everything except for every $1 \leqslant i \leqslant m$, where x_i is mapped to p_i.

We extended the notion of an environment $\mathcal{E}$ in [22, 23]. An environment is a mapping from name variables to names. Suppose we are given a partial ordering schedule $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ and a finite subset $\omega \subseteq \Omega$. We shall define what it means to say that, in Ψ, TPOL formula S is true from ω under environment $\mathcal{E}$, written as $\Psi : \omega \models_{\mathcal{E}} S$.

For every ω, defined in Ψ, and $\mathscr{E}$:

$\Psi:\omega \models_{\mathscr{E}} true$

$\Psi:\omega \models_{\mathscr{E}} e[x]+c \leqslant f[y]+d$ iff $\Gamma \models \Lambda(e,\mathscr{E}(x))+c \leqslant \Lambda(f,\mathscr{E}(y))+d$

$\Psi:\omega \models_{\mathscr{E}} \neg S_1$ iff it is not the case that $\Psi:\omega \models_{\mathscr{E}} S_1$

$\Psi:\omega \models_{\mathscr{E}} S_1 \vee S_2$ iff $\Psi:\omega \models_{\mathscr{E}} S_1$ or $\Psi:\omega \models_{\mathscr{E}} S_2$

$\Psi:\omega \models_{\mathscr{E}} \triangle\{[x_1],\ldots,[x_m]\}S_1$ iff for some $p_1,\ldots,p_m \in \Theta$ s.t.

- for every $a_i \in \omega$ and $b_j \in \text{start}(\Psi,\{\Lambda[p_1],\ldots,\Lambda[p_m]\})$, $\Gamma \models a_i \leqslant b_j$; and
- $\Psi:\text{start}(\Psi,\{\Lambda[p_1],\ldots,\Lambda[p_m]\}) \models_{\mathscr{E}[x_i \leftarrow p_i \mid 1 \leqslant i \leqslant m]} S_1$

We say partial ordering schedule Ψ is a *model* of a well-formed TPOL formula S or that Ψ *satisfies* S, written as $\Psi \models S$, iff $\Psi:\{0_1\} \models_{\emptyset} S$. A TPOL formula S is *satisfiable* iff there is a model for S.

There are two things that we should note here. First, because we only have precedence relations in our literals, there is no equality for objects in TPOL. However, it is possible to use TPOL to determine if two transactions have the same event occurrence times for event types. Second, since our precedence relation is true only when both events are defined, something like $e[a] \leqslant e[a]$ is equivalent to the test for existence of event $e[a]$. Indeed, in the following, we shall use this type of literal to test for event existences.

The general satisfiability problem of TPOL formulas is undecidable. A proof can be obtained by modifying the undecidability proof of the AREL sentence satisfiability problem in [23]. In Sections 4–6, we will consider the possibility of using restrictions on TPOL formulas to define TPOL sublanguages with decidable satisfiability problems. TPOL_1 allows only one event type in the formulas while TPOL_2 allows at most two. The third sublanguage, TPOL_{SBS}, restricts the span of each transaction, which is a similar idea to the BOIO restriction [23].

3. The physical world specified by TPOL

In this section, we give the motivation for the design considerations of TPOL and its models. We also present arguments for all those design decisions. We divide our explanation into three subsections. Section 3.1 explains the physical meaning of partial-ordering schedules. Section 3.2 explains the meaning of the two new modal operators, $\triangle, \triangledown$. Section 3.3 gives several examples to show how TPOL can be used in the description of high-level behaviors in real-time transaction systems.

3.1. The physical world specified by partial-ordering schedules

The behaviors (models) of TPOL formulas are partial-ordering schedules. Suppose we have a partial-ordering schedule $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ for a TPOL formula S. Then H_S is the number of clocks used in Ψ. Each local clock reading $a_i \in \Omega$ represents an

event observed at reading $a \in \mathcal{N}$ with respect to clock i. Due to the partial-ordering nature of distributed processing and the possibility of different clock granularities and jitter, the concept of local clock readings lays a convenient basis for further research on the description and verification of distributed systems. Thus, the addition of clock readings of different sites is meaningless in our systems since clock readings are treated as temporal marks instead of numerical values.

Note that, given integers a and $1 \leqslant i \leqslant H_S$, we allow a_i to be undefined in a partial-ordering schedule (i.e. $a_i \notin \Omega$). This is a plausible assumption since, in a clock-synchronized environment such as the one described in [13], clocks may advance their readings by an unbounded amount, both noncontinuously and instantaneously. In such a situation, those skipped reading values never exist.

Γ is defined as a partial-ordering relation. Conceptually, a pair $(a_i, b_j) \in \Gamma$ iff in Ψ clock i reads a no later than clock j reads b. For example, $\Gamma \models 5_2 \prec 3_1$ iff clock 2 reads five earlier than clock 1 reads three in Γ. Note that, under this definition, our partial-ordering relations $(\leqslant, \geqslant, \prec, \succ, \asymp)$ degenerate to traditional integer inequalities when there is only one clock in the system. Intuitively, in Lamport's partial-ordering computing models [13], $a_i \prec b_j$ can be interpreted as meaning that when clock i reads a, a message is sent out form site i, and that message is received at site j when clock j reads b.

E_S is the set of event types used in S. An event type represents a set of temporal marks, i.e. events, with a specific property and may have any number of occurrences in Ψ. Examples of events are the ticking of a clock, toggling of a Boolean variable, sending or reception of messages, and detection of a hardware interrupt. The events of a type are all instantaneous and can only be measured with respect to a specific local clock. That is, we assume that each event of the same type can only be observed at a particular site in a distributed system.

Θ is a set of names (also called *identifiers* in the literature) used to distinguish among events of the same type. In such a situation, the computer had better not mix up these many names. It has been a common practice to give each user a unique name. Users can be real people or system processes. For example, in a multiuser system, we may have users FARN, FRANK, and PRINTER trying to access the same data file at the same instant.

We can also use names to relate events of different types. Events of different types labeled with the same name represent a special activity invoked by, perhaps, a single user. Such an activity corresponds to the prevailing concept of a *transaction*. In Section 2, the transaction with name p was denoted by $[p]$. The local clock readings of events in transactions are defined by Λ. The correct execution of a transaction relies on the proper timing consortium among the internal events. Also, different transactions may interact with each other. TPOL stands as a tool for unified specification of both the internal consortium and peer transaction interaction.

Example 4. Shown in Fig. 1 is a partial-ordering schedule segment for 2 local clocks and event type set $\{e, f, g\}$. Solid arcs represent partial-ordering relations between

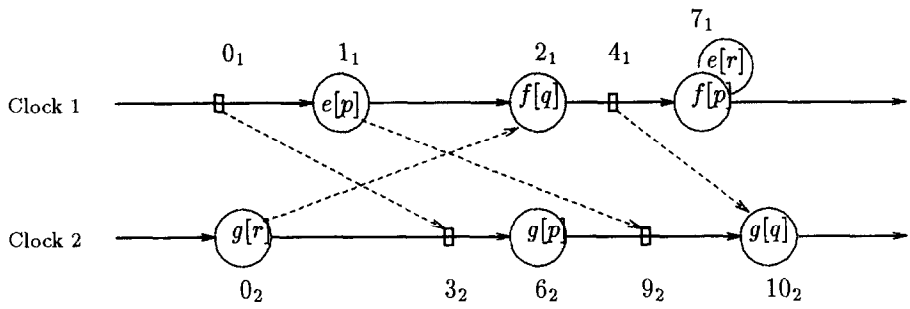

Fig. 1. A partial-ordering schedule segment.

events specific to the same local clock. Dashed arcs represent partial-ordering relations between events specific to different local clocks and may be interpreted as message communications. Also three names, p, q, and r, and the existing events of transactions $[p], [q]$, and $[r]$ are all shown in the segment.

Thus, for transaction $[p]$, $\Lambda(e, p) = 1_1$, $\Lambda(f, p) = 7_1$, and $\Lambda(g, p) = 6_2$. For transaction $[q]$, $\Lambda(e,q) = \perp$, $\Lambda(f,q) = 2_1$, and $\Lambda(g,q) = 10_2$. For transaction $[r]$, $\Lambda(e,r) = 7_1$, $\Lambda(f,r) = \perp$, and $\Lambda(g,r) = 0_2$. Since, according to Fig. 1, $\Lambda \models 0_1 \leqslant 3_2$, we also have $\Lambda \models \Lambda(e, p) - 1 \leqslant \Lambda(g,r) + 3$.

Note that $1_1 \leqslant 6_2$ is not satisfied in the partial-ordering schedule because the pair $(1_1, 6_2)$ is not recorded in Γ. Also note that $1_1 \leqslant 6_1$ is not true because event 6_1 does not exist in the partial-ordering schedule.

3.2. *The physical meaning of* $\triangle$ *and* ∇

In TPOL, we specifically define a new modal operator $\triangle$ to characterize the interactions among peer transactions. $\triangle$ is the "*eventually*" modal operator in event-based temporal logic [23] with built-in semantics for the description of distributed computing. Formally, $\triangle\{[x_1], \ldots, [x_m]\}S_1$ is meant to assert "from now on eventually there will be transactions $[x_1], \ldots, [x_m]$ which make S_1 true". Intuitively, when we assert $\Psi : \omega \models_{\mathscr{E}} \triangle\{[x_1], \ldots, [x_m]\}S_1$, it specifies a reactive mechanism of m transactions, say $[p_1], \ldots, [p_m]$, such that the reaction is

- posted by events in ω,
- is served when a message is received from each event in ω by each event in $\Lambda[p_1] \cup \cdots \cup \Lambda[p_m]$, and
- is fulfilled when S_1 is satisfied with the interpretation of $x_1 \to p_1, \ldots, x_m \to p_m$.

On the other hand, $\nabla\{[x_1], \ldots, [x_m]\}S_1$ is meant to assert "from now on forever for every transactions $[x_1], \ldots, [x_m]$, S_1 must be true". An interesting thing happens when we have nested quantification. For example, we may have a formula

$$\nabla\{[x_1], [x_2]\}(\text{COLLISION}^{x_1, x_2} \to \triangle\{[y]\}\text{AMBULANCE}^{x_1, x_2, y})$$

where $\text{COLLISION}^{x_1, x_2}$ is a TPOL formula with free variable x_1, x_2 and $\text{AMBULANCE}^{x_1, x_2, y}$ is another TPOL formula with free variable x_1, x_2, y. Intuitively,

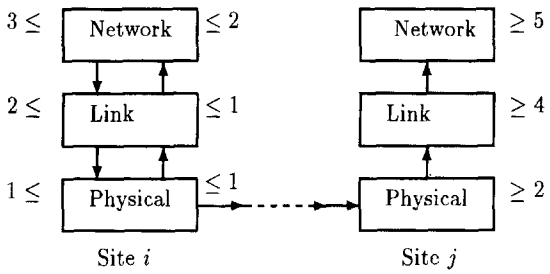

Fig. 2. Timing behavior of network layer communication.

the formula states the safety mechanism that for every collision of two cars x_1, x_2, there is an ambulance y which provides service specified in AMBULANCE^{x_1,x_2,y} to x_1, x_2. The message-passing spirit of TPOL can be revealed when we look into the formal semantics of such a formula. By interpreting the precedence relation as message-passing (or causality), we find that the formula actually means that for every two cars $[p_1], [p_2]$ colliding with each other, after a message is received from each event in start$(\Psi, \{\Lambda[p_1], \Lambda[p_2]\})$ by ambulance $[q]$, the service will be provided according to AMBULANCE^{x_1,x_2,y}.

3.3. Examples of TPOL

We will use the following two examples to conveniently specify two computer behaviors and to justify our definition of TPOL.

Example 5. Suppose we have a network connecting sites i and j, where $1 \leqslant i, j \leqslant H_S$, in a distributed system as shown in Fig. 2. We have three protocol layers. The number by the upper-left (upper-right) corner of each layer box is an estimation of the time required to move down (up) a message through the corresponding layer. For example, to pass a message down through this network layer at site i takes at least 3 time units according to clock i. For each message passing from site i to site j, there are three events involved from the viewpoint of the network layer. They are $Network\_Send^{(i)}$ for sending a message at the network layer at site i, $Network\_Sent^{(i)}$ for the acknowledgment from physical to network layer at site i that the message has been sent, and $Network\_Received^{(j)}$ for the reception of a message at the network layer at site j. For each transaction between each two sites, the two partial-ordering relations

$$\nabla\{[x]\}\begin{pmatrix} Network\_Send^{(i)}[x] + 6 \preccurlyeq Network\_Received^{(j)}[x] - 11 \\ \wedge\ Network\_Sent^{(i)}[x] - 4 \preccurlyeq Network\_Received^{(j)}[x] - 11 \end{pmatrix}$$

describe the timing behavior of the message passing at the network layer between two sites in a very succinct way. Note that in the formula $E_S = \{Network\_Send^{(i)}, Network\_Received^{(j)}, Network\_Sent^{(i)}\}$.

Example 6. In a stack system with two event types, *Push* and *Pop*, each item pushed into the stack corresponds to a transaction. The behavior of the stack can be described by the following formula:

$$\nabla\{[x_1],[x_2]\}\left(Push[x_1] \asymp Push[x_2] \leftrightarrow \left(\begin{array}{l}\left(\begin{array}{r}\neg Pop[x_1] \asymp Pop[x_1]\\ \wedge\ \neg Pop[x_2] \asymp Pop[x_2]\end{array}\right)\\ \\ \vee\ Pop[x_1] \asymp Pop[x_2]\end{array}\right)\right)$$

$$\wedge\ \nabla\{[x_1],[x_2]\}\,(Pop[x_1] \asymp Pop[x_2] \rightarrow Push[x_1] \asymp Push[x_2])$$

$$\wedge\ \nabla\{[x]\}\,(Push[x] \asymp Push[x] \wedge \neg Push[x] \gtrsim Pop[x])$$

$$\wedge\ \nabla\{[x_1],[x_2]\}((Push[x_1] \prec Push[x_2] \wedge Pop[x_1] \asymp Pop[x_1]) \rightarrow Pop[x_2] \prec Pop[x_1]))$$

The first two lines say that two transactions have the same value iff they have the same *Push* or *Pop* occurrence time. The third line says that, for each transaction, the *Push* event exists, and that *Pop* is always after the *Push*. Note here that we use $Push[x] \asymp Push[x]$ to test for the existence of $Push[x]$. The fourth line says first in last out.

4. TPOL$_1$, a sublanguage with only one event type

TPOL$_1$ is the sublanguage of TPOL with only one event type allowed.

Theorem 1. *The satisfiability problem of* TPOL$_1$ *is* EXPSPACE-*hard.*

Proof. Given an EXPSPACE Turing machine halting problem instance with tape input of length n, we shall use an interval of $8 + n + 10.2^n$ time units to encode an *instantaneous description* (*ID*) [7] of the machine, as illustrated in Fig. 3. We call such a partial-ordering schedule segment an *ID frame*. For convenience of discussion, we label each time slot of the partial-ordering schedule with a Boolean value such that, when there is an event, the value is 1; otherwise, it is 0. Of each ID frame, the first five time slots have five e events and mark the beginning of the encoding of an instantaneous description. Thus, the beginning of each ID frame should start with Boolean string 11111. Slots 7 to $8+n$ are used to encode the current state of M. From slot $9 + n$ to slot $8 + n + 10 \cdot 2^n$, every 10 time units encode a tape cell, and this is called a *TC frame*. For each tape cell, the ten time units should start with Boolean string 11011.

Note that, due to our design, except for the Boolean strings which start ID frames and tape cells, in the ID frames we have Boolean value 0 at every other time slot. Thus, there is no confusion in recognizing the appearance of an ID frame starting Boolean strings and a tape cell starting Boolean strings.

Two events that are $O(2^n)$ time units apart can be related by a partial-ordering relation with an $O(n)$-bit timing constant. With the syntax and semantics of TPOL$_1$,

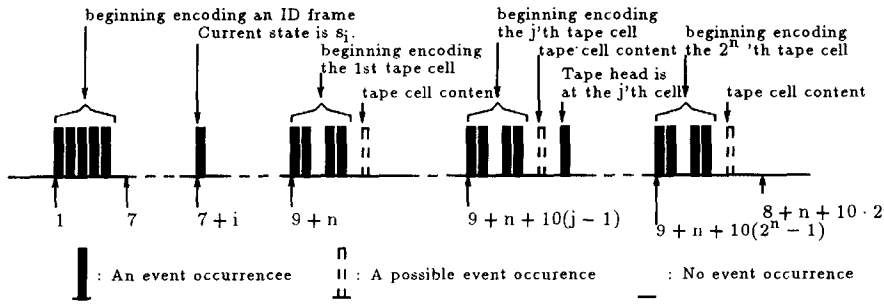

Fig. 3. An ID frame to encode an instantaneous description.

we may define various structures to construct the computation of EXPSPACE Turing machines. For example, we may use IDstart(p) as shorthand for

$$\triangle\{[x_2],[x_3],[x_4],[x_5]\}\begin{pmatrix} e[p] \asymp e[x_2]-1 \wedge e[p] \asymp e[x_3]-2 \\ \wedge\ e[p] \asymp e[x_4]-3 \wedge e[p] \asymp e[x_5]-4 \end{pmatrix}$$

which means that $e[p]$ is the start of an ID frame. We may also use TCstart(p) as shorthand for

$$\left(\triangle\{[x_2],[x_3],[x_4]\}\begin{pmatrix} e[p] \asymp e[x_2]-1 \\ \wedge\ e[p] \asymp e[x_3]-3 \\ \wedge\ e[p] \asymp e[x_4]-4 \end{pmatrix}\right) \wedge \left(\nabla\{[x_5]\}\begin{pmatrix} e[p] \not\asymp e[x_5]-2 \\ \wedge\ e[p] \not\asymp e[x_5]-5 \\ \wedge\ e[p] \not\asymp e[x_5]-7 \\ \wedge\ e[p] \not\asymp e[x_5]-9 \end{pmatrix}\right)$$

which means that $e[p]$ is the start of a tape cell frame. We may then define the shorthand TCinID(p_1, p_2) as

$$e[p_1]+9+n \preccurlyeq e[p_2] \ \wedge\ e[p_1]+9+n+10\cdot(2^n-1) \succcurlyeq e[p_2]$$

which means that if there are an ID frame beginning with $e[p_1]$ and a TC frame beginning with $e[p_2]$, then the corresponding TC frame is in the ID frame. We also use Cell0(p) as shorthand for

$$\text{TCstart}(p) \wedge \nabla\{[y]\}e[p]+6 \not\asymp e[y]$$

which encodes that $e[p]$ is the start of a tape cell, and that the content of the corresponding tape cell is 0. Then the ID succession, tape cell content changes, and tape head movement can all be described by proper combination of these structures and can be used to prove the theorem.

We define State(p, i) as shorthand for

$$\begin{pmatrix} \text{IDstart}(p) \\ \wedge\ \triangle\{[y_1]\}(e[p]+7+i \asymp e[y_1]) \\ \wedge\ \nabla\{[y_2]\}(e[p]+4 \succcurlyeq e[y_2] \vee e[p]+7+i \asymp e[y_2] \vee e[p]+9+n \preccurlyeq e[y_2]) \end{pmatrix}$$

which encodes that if there is an ID frame which starts at $e[p]$, then the corresponding ID is at state s_i. We define Headpos(p) as shorthand for

$$\text{TCstart}(p) \wedge \triangle\{[y_1]\}e[p] + 8 \asymp e[y_1]$$

which encodes that an TC frame starts at $e[p]$, and that the tape head of the corresponding ID is at the tape cell represented by the TC frame starting at $e[p]$. Similarly, we can define shorthand Cell1(p) to detect if the tape cell content is 1.

The formula $S \in \text{TPOL}_1$ which encodes the successive ID's of M is a conjunction $S_1 \wedge S_2 \wedge S_3 \wedge S_4 \wedge S_5$. In the following, we shall list the requirements on S and encode them in S_1, S_2, S_3, S_4, and S_5, respectively.

- *ID frames encode successive state transition of M.* We will use a typical transition to show how we encode this requirement. Suppose we have a transition which at state s_i goes to state s_j if the tape head reads 0. The following formula encodes this state transition:

$$\nabla\{[y_1],[y_2]\}\left(\begin{pmatrix}\text{State}(y_1,i)\\ \wedge\ \text{Headpos}(y_2)\\ \wedge\ \text{TCinID}(y_1,y_2)\\ \wedge\ \text{Cell0}(y_2)\end{pmatrix} \rightarrow \triangle\{[x]\}\begin{pmatrix}\text{State}(x,j)\\ \wedge\ e[y_1]+8+n+10\cdot 2^n \asymp e[x]\end{pmatrix}\right)$$

  S_1 should encode all the state transition information of M in this fashion.
- *Tape cell contents change according to the execution of M.* We will use a typical transition to show how we encode this requirement. Suppose we have a transition which at state s_i writes value 1 if the tape head reads 0. The following formula encodes this state transition:

$$\nabla\{[y_1],[y_2]\}\left(\begin{pmatrix}\text{State}(y_1,i)\\ \wedge\ \text{Headpos}(y_2)\\ \wedge\ \text{TCinID}(y_1,y_2)\\ \wedge\ \text{Cell0}(y_2)\end{pmatrix} \rightarrow \triangle\{[x]\}\begin{pmatrix}\text{Cell1}(x)\\ \wedge\ e[y_2]+8+n+10\cdot 2^n \asymp e[x]\end{pmatrix}\right)$$

  Also, we are aware that a tape cell's content should not change when the tape head is not on it:

$$\nabla\{[y]\}\left(\begin{pmatrix}\text{TCstart}(y)\\ \wedge\ \neg\text{Headpos}(y)\end{pmatrix} \rightarrow \triangle\{[x]\}\begin{pmatrix}\text{TCstart}(x)\\ \wedge\ e[y_2]+8+n+10\cdot 2^n \asymp e[x]\\ \wedge\ (\text{Cell1}(y) \leftrightarrow \text{Cell1}(x))\end{pmatrix}\right)$$

  S_2 should encode all the state transition information of M in this fashion.
- *Tape head moves according to the execution of M.* This part is simple but tedious to express. We choose to list the items which should be taken care of in English.
  - In the next state, the tape head will not stay at the same tape cell as in the current state:

$$\nabla\{[y]\}(\text{Headpos}(y) \rightarrow \nabla\{[x]\}e[y] + 16 + n + 10\cdot 2^n \not\asymp e[x])$$

- If the tape head is at the first cell, then the tape head can only move right.
- If the tape head is at the last cell, then the tape head can only move left.
- If a tape cell is at least two cells away from the tape head in the current state, then it will not be read by the tape head in the next state.
- If a tape cell is one cell to the right of the tape head in the current state, then it will be read by the tape head in the next state if the next transition moves the head right. The symmetric case when the tape cell is one cell to the left must also be true.

S_3 should encode all these requirements.

- *Initially all the tape cells contain binary zeros*:

$$S_4 \equiv \triangle\{[z]\}\left(\begin{array}{l} \mathrm{State}(z,0) \\ \wedge\ \nabla\{[y]\}e[z] \preccurlyeq e[y] \\ \wedge \left(\begin{array}{l} \triangle\{[y_1]\}(\mathrm{Headpos}(y_1) \wedge \mathrm{Cell0}(y_1) \wedge e[z]+8+n \asymp e[y_1]) \\ \wedge\ \nabla\{[y_2]\}\left(\begin{array}{l} \left(\begin{array}{l} \mathrm{TCstart}(y_2) \\ \wedge\ e[z]+9+n \preccurlyeq e[y_2] \\ \wedge\ e[z]+9+n+10\cdot(2^n-2) \succcurlyeq e[y_2] \end{array}\right) \\ \rightarrow \triangle\{[x]\}\left(\begin{array}{l} \mathrm{TCstart}(x) \\ \wedge\ \mathrm{Cell0}(x) \\ \wedge\ \neg\mathrm{Headpos}(x) \\ \wedge\ e[y_2]+10 \asymp e[x] \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right)$$

In S_4, we identify a particular event $e[z]$ which is the first event in the partial-ordering schedule. Then we define the first ID frame with respect to $e[z]$.

- *The halting problem of M is to detect whether s_n is reachable*:

$$S_5 \equiv \triangle\{[x]\}\mathrm{State}(x,n)$$

It can be shown that the reduced formula can be generated in PTIME. This way, the EXPSPACE-hardness of the satisfiability problem of TPOL_1 is proven. □

5. TPOL_2, a sublanguage with at most two event types

TPOL_2 is the sublanguage of TPOL with at most two event types allowed.

Theorem 2. *The single-clock* TPOL_2 *satisfiability problem is undecidable.*

Proof. We shall reduce the *post's correspondence problem* (*PCP*) [6, 7, 16] to the satisfiability problem of TPOL_2. An instance of PCP [7] consists of two lists $\ddot{V} = v_1, \ldots, v_k$ and $\ddot{W} = w_1, \ldots, w_k$ of strings over some alphabet Σ. This instance of PCP has a solution iff there is any sequence of integers $i_1, \ldots, i_n$ with $n \geqslant 1$ such that $v_{i_1} v_{i_2} \ldots v_{i_n} = w_{i_1} w_{i_2} \ldots w_{i_n}$. For convenience, we call $v_{i_1} v_{i_2} \ldots v_{i_n}$ a solution string of the PCP instance.

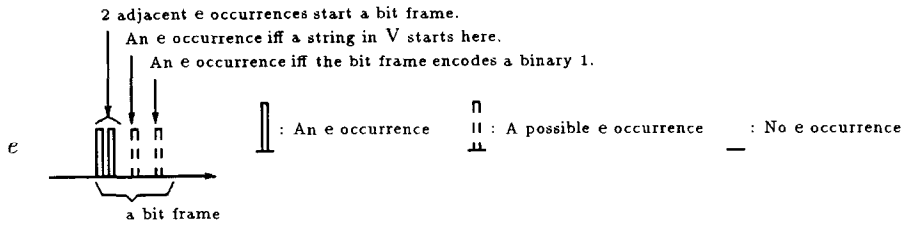

Fig. 4. A PCP bit frame.

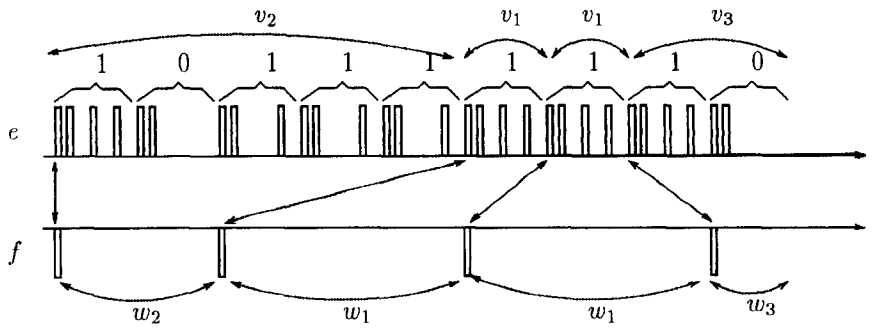

Fig. 5. A PCP solution string.

We shall use event types e, f which share the same clock. Each seven time unit interval, called a *bit frame*, emulates a bit in the solution string of the PCP instance. The e-events in a bit-frame encode the necessary information in order to tell

- which e-events along the partial-ordering schedule start bit frames;
- which bit frames are the starts of strings in $\ddot{V}$; and
- whether a bit frame encodes a zero or a one.

Each bit frame starts with two consecutive e events occupying the first and second time slots, as shown in Fig. 4. The third, fifth, and seventh time slots must not have any event of type e. The fourth one has an e event iff the bit frame encodes the starting bit of a string in $\ddot{V}$. A bit frame that encodes the start of a string in $\ddot{V}$ is called a *$\ddot{V}$-frame*. The sixth one has an e event iff the bit frame encodes a binary 1.

None of the e events has corresponding f events unless it is the first e event in a $\ddot{V}$-frame. For every name p, if $f[p]$ exists, then $f[p]$ coincides with the first e event in a bit frame which encodes the starting bit of a string in $\ddot{W}$ and is called a $\ddot{W}$-frame.

We will use the example in [7] to illustrate this encoding scheme. Suppose we are given the following PCP instance:

$$v_1 = 1, \quad v_2 = 10111, \quad v_3 = 10$$
$$w_1 = 111, \quad w_2 = 10, \quad w_3 = 0$$

One solution string is $v_2 v_1 v_1 v_3 = w_2 w_1 w_1 w_3 = 101111110$. The partial-ordering schedule segment which encodes this solution string is in Fig. 5. We shall reduce PCP to TPOL_2

formula for a system with only one local clock. Suppose that the only allowed two event types are e and f. We assume that the strings are in binary representation. The reduction utilizes the unlimited cross reference capability between $e[p]$ and $f[p]$ for any name p.

We need the following shorthand to construct the encoding formulas. BFstart(p) is shorthand for

$$\triangle\{[x]\}e[p]+1 \asymp e[x]$$

which is true iff $e[p]$ marks the start of a bit frame. Value($p,0$) is shorthand for

$$\text{BFstart}(p) \wedge \nabla\{[x]\}e[p]+5 \not\asymp e[x]$$

which is true iff $e[p]$ marks the start of a bit frame and the bit is binary zero. Value($p,1$) is shorthand for

$$\text{BFstart}(p) \wedge \triangle\{[x]\}e[p]+5 \asymp e[x]$$

which is true iff $e[p]$ marks the start of a bit frame and the bit is binary one.

VSstart(p) is shorthand for

$$\text{BFstart}(p) \wedge \triangle\{[x]\}e[p]+3 \asymp e[x]$$

which is true iff $e[p]$ marks the start of a $\ddot{V}$-frame.

Given a binary string $v=\alpha_1\alpha_2\ldots\alpha_h$ in $\ddot{V}$, Vstring(p,v) is shorthand for

$$\triangle\begin{Bmatrix}[x_2],\\ \vdots\\ [x_h],\\ [x_{h+1}]\end{Bmatrix}\begin{pmatrix} & \text{BFstart}(p)\wedge\text{Value}(p,\alpha_1)\\ \wedge & \text{BFstart}(x_2)\wedge e[p]+7\asymp e[x_2]\wedge\text{Value}(x_2,\alpha_2)\wedge\neg\text{VSstart}(x_2)\\ \wedge & \text{BFstart}(x_3)\wedge e[x_2]+7\asymp e[x_3]\wedge\text{Value}(x_3,\alpha_3)\wedge\neg\text{VSstart}(x_3)\\ \ldots & \ldots\ldots\ldots\\ \wedge & \text{BFstart}(x_h)\wedge e[x_{h-1}]+7\asymp e[x_h]\wedge\text{Value}(x_h,\alpha_h)\wedge\neg\text{VSstart}(x_h)\\ \wedge & \text{VSstart}(x_{h+1})\wedge e[x_h]+7\asymp e[x_{h+1}]\end{pmatrix}$$

and is true iff $e[p]$ is the starting e event of the encoding partial-ordering schedule for v.

WSstart(p) is shorthand for

$$\text{BFstart}(p) \wedge \triangle\{[x]\}e[p] \asymp f[x]$$

which is true iff $e[p]$ marks the start of a $\ddot{W}$-frame.

Given a binary string $w=\beta_1\beta_2\ldots\beta_k$ in $\ddot{W}$, Wstring(p,w) is shorthand for

$$\triangle\left\{\begin{matrix}[x_2],\\ \vdots\\ [x_k],\\ [x_{k+1}]\end{matrix}\right\}\left(\begin{matrix} & \mathrm{BFstart}(p)\wedge\mathrm{Value}(p,\beta_1)\\ \wedge & \mathrm{BFstart}(x_2)\wedge e[p]+7\asymp e[x_2]\wedge\mathrm{Value}(x_2,\beta_2)\wedge\neg\mathrm{WSstart}(x_2)\\ \wedge & \mathrm{BFstart}(x_3)\wedge e[x_2]+7\asymp e[x_3]\wedge\mathrm{Value}(x_3,\beta_3)\wedge\neg\mathrm{WSstart}(x_3)\\ & \cdots\cdots\cdots\\ \wedge & \mathrm{BFstart}(x_k)\wedge e[x_{k-1}]+7\asymp e[x_k]\wedge\mathrm{Value}(x_k,\beta_k)\wedge\neg\mathrm{WSstart}(x_k)\\ \wedge & \mathrm{WSstart}(x_{k+1})\wedge e[x_k]+7\asymp e[x_{k+1}]\end{matrix}\right)$$

and is true iff $f[p]$ marks the start of the encoding partial-ordering schedule for w.

The encoding formula in TPOL_2 must satisfy the following three requirements.

- *Initialization of the encoding partial-ordering schedule*:

$$S_1\equiv\triangle\{[x]\}\left(\begin{matrix} & \neg(f[x]\asymp f[x])\\ \wedge & \neg\mathrm{VSstart}(x)\\ \wedge & \nabla\{[y]\}\neg(e[y]\prec e[x]\vee f[y]\prec e[x])\\ \wedge & \left(\begin{matrix} & (\mathrm{Vstring}(x,v_1)\wedge\mathrm{Wstring}(x,w_1))\\ \vee & (\mathrm{Vstring}(x,v_2)\wedge\mathrm{Wstring}(x,w_2))\\ & \cdots\cdots\cdots\\ \vee & (\mathrm{Vstring}(x,v_k)\wedge\mathrm{Wstring}(x,w_k))\end{matrix}\right)\end{matrix}\right)$$

- *The correspondence between event types e and f*:

$$S_2\equiv\left(\begin{matrix} & (\nabla\{[x]\}\mathrm{VSstart}(x)\rightarrow f[x]\asymp f[x])\\ \wedge & (\nabla\{[x]\}\mathrm{VSstart}(x)\wedge\triangle\{[y]\}(\mathrm{BFstart}(y)\wedge f[x]\asymp e[y]))\\ \wedge & (\nabla\{[x_1],[x_2]\}f[x_1]\preccurlyeq f[x_2]\rightarrow e[x_1]\preccurlyeq e[x_2])\end{matrix}\right)$$

- *Succession of the strings from the two lists*:

$$S_3\equiv\left(\begin{matrix} & \bigwedge_{1\leqslant i\leqslant k}\nabla\{[x_1]\}\left(\left(\begin{matrix}\mathrm{VSstart}(x_1)\\ \wedge\ \mathrm{Vstring}(x_1,v_i)\end{matrix}\right)\right.\\ & \qquad\left.\rightarrow\nabla\{[x_2]\}\left(\left(\begin{matrix}\mathrm{VSstart}(x_2)\\ \wedge\ f[x_1]\asymp e[x_2]\end{matrix}\right)\rightarrow\mathrm{Wstring}(x_2,w_i)\right)\right)\\ \wedge & \bigwedge_{1\leqslant i\leqslant k}\nabla\{[x_2]\}\left(\left(\begin{matrix}\mathrm{VSstart}(x_2)\\ \wedge\ \mathrm{Wstring}(x_2,w_i)\end{matrix}\right)\right.\\ & \qquad\left.\rightarrow\nabla\{[x_1]\}\left(\left(\begin{matrix}\mathrm{VSstart}(x_1)\\ \wedge\ f[x_1]\asymp e[x_2]\end{matrix}\right)\rightarrow\mathrm{Vstring}(x_1,v_i)\right)\right)\end{matrix}\right)$$

- *The detection of a solution*:

$$S_4\equiv\triangle\{[x]\}f[x]\asymp e[x]$$

The key idea of this reduction is in S_3 when we require that for some $1 \leqslant i \leqslant k$, v_i starts at $e[x_1]$ and w_i starts at $f[x_1] \asymp e[x_2]$. Given a PCP instance, there is a solution for this instance iff $S_1 \wedge S_2 \wedge S_3 \wedge S_4$ is satisfiable. □

6. $\mathrm{TPOL}_{\mathrm{SBS}}$: a sublanguage for synchronized bounded-span transactions

Theorem 2 also establishes the undecidability of the general TPOL satisfiability problem and implies that by restricting the number of event types used, perhaps no reasonably practical sublanguages of TPOL can be obtained. Here we propose another restriction on the syntax of TPOL formulas, the SBS (*synchronized bounded span*) restriction, which naturally maps to the following two physical requirements which may be imposed on any distributed system design:

- Each transaction must be short, and its events must strongly correlate with one another. The short transaction assumption is adequate for classical transaction processing theory, which deals with activities like depositing and withdrawing in teller machines.
- The different local clocks in the distributed system must strongly synchronize with one another. In fact, we ask that, for each local clock, there is a predefined maximum period for successive synchronization with all the other local clocks.

Under these two strong conditions, we shall show that a sequentialization scheme for each satisfying partial-ordering schedule, with respect to a given $\mathrm{TPOL}_{\mathrm{SBS}}$ formula, can be proposed. Then under the sequentialization scheme, the technique of tableau construction in real-time temporal logic [1, 24, 22] can be employed to define an algorithm for the $\mathrm{TPOL}_{\mathrm{SBS}}$ satisfiability problem.

Definition 2 (*Synchronized bounded span restriction*). A TPOL formula S satisfies the *synchronized bounded span restriction* if there is a nonnegative integer constant B_S such that S can be represented by the conjunction

$$
\begin{aligned}
S_* \wedge \Bigg(& \bigwedge_{e,f \in E_{S_*}} \nabla \{[x_1],[x_2]\} \Big(\Big(e[x_1] + d^{\langle i(e) \rangle} \leqslant f[x_2] \\
& \quad \rightarrow \bigwedge_{e',f' \in E_{S_*}} ((e'[x_1] \neq \perp \wedge f'[x_2] \neq \perp) \rightarrow e'[x_1] \leqslant f'[x_2]) \Big) \Big) \\
& \wedge \triangle \{[z_1], \ldots, [z_{H_S}]\} \Big(\Big(\bigwedge_{1 < i \leqslant H_S} \pi_1[z_1] \asymp \pi_i[z_i] \Big) \\
& \qquad \wedge \nabla \{[x]\} \bigwedge_{e \in E_{S_*}} (e[x] \neq \perp \rightarrow \pi_1[z_1] \leqslant e[x]) \Big) \\
\wedge & \bigwedge_{1 \leqslant i,j \leqslant H_S} \nabla \{[x_1],[x_2]\} \Big(\pi_i[x_1] \asymp \pi_j[x_2] \rightarrow \triangle \{[y_1],[y_2]\} \\
& \Big(\Big(\begin{array}{l} \pi_i[x_1] + d^{\langle i \rangle} \succ \pi_i[y_1] \wedge \pi_i[x_1] \prec \pi_i[y_1] \\ \wedge \pi_j[x_2] + d^{\langle j \rangle} \succ \pi_j[y_2] \wedge \pi_j[x_2] \prec \pi_j[y_2] \end{array} \Big) \wedge \pi_i[y_1] \asymp \pi_j[y_2] \Big) \Big) \Bigg)
\end{aligned}
$$

where S_* is an arbitrary well-formed (no free variables) TPOL formula, and $\pi_1, \ldots, \pi_{H_S} \notin E_{S_*}$, with $\iota(\pi_i) = i$ for $1 \leqslant i \leqslant H_S$, are special event types used to strongly synchronize the different sites of the system. Also, for each $1 \leqslant i \leqslant H_S$, $d^{\langle i \rangle}$ is a natural number $\leqslant B_S$. B_S is called the *transaction span bound* of this formula.

Anything in a TPOL_{SBS} formula besides part S_*, which is the true transaction specification of interest to users, is conveniently called an *SBS restriction*. Note that the way we present the SBS restriction demonstrates that TPOL_{SBS} is indeed a subclass of TPOL. In practice, language designers can just ask users to provide values of $d^{\langle 1 \rangle}, d^{\langle 2 \rangle}, \ldots, d^{\langle H_S \rangle}$ instead of the three SBS restriction conjuncts. As we shall soon see, this is partly because $\pi_1, \ldots, \pi_{H_S} \notin E_{S*}$ are merely used to strongly synchronize the different local clocks and do not participate in the transaction execution described in S_*.

Given a formula in TPOL_{SBS}, we shall see that the SBS restriction can be enforced during tableau construction by using simple checking routines on tableau nodes. Formula S_* is the true behavior description of transactions, and the remaining three conjuncts are the SBS restrictions. The first conjunct of the SBS restrictions

$$\bigwedge_{e,f \in E_S} \nabla \{[x_1],[x_2]\} \left(\left(e[x_1] + d^{\langle \iota(e) \rangle} \leqslant f[x_2] \right. \right.$$

$$\left. \left. \rightarrow \bigwedge_{e',f' \in E_S} ((e'[x_1] \neq \perp \wedge f'[x_2] \neq \perp) \rightarrow e'[x_1] \leqslant f'[x_2]) \right) \right)$$

means that, given two transactions $[p],[q]$, if an event in $[p]$ happens well ahead of another event in $[q]$, then $[p]$ precedes $[q]$ event by event. In particular, it says that for each clock, an interval longer than B_S cannot be covered by a single transaction.

The second and third conjuncts of the SBS restrictions impose strict synchronization among the sites. Specifically, the second conjunct says that $\pi_1[z_1], \ldots, \pi_{H_S}[z_{H_S}]$ are the starting events of the partial-ordering schedule. The third conjunct says that, within a $d^{\langle i \rangle}$ time unit interval, clock i must synchronize with all other clocks at least once.

The EXPSPACE-hardness of the TPOL_1 satisfiability problem directly implies the EXPSPACE-hardness of TPOL_{SBS} satisfiability problem.

Corollary 3. *The satisfiability problem of* TPOL_{SBS} *is* EXPSPACE-*hard.*

Proof. We note that TPOL_{SBS} is a superclass of TPOL_1 because the conjunction of TPOL_{SBS} given in Definition 2, except for S_*, becomes a tautology when $|E_{S_*}| = 1$. □

In the following, we shall present the design of a tableau-based algorithm for the TPOL_{SBS} satisfiability problem. Given an $S \in \text{TPOL}_{\text{SBS}}$, the tableau constructed here will be a finite directed graph such that a path in the graph roughly corresponds to a partial-ordering schedule segment for S. In Section 6.1, we shall first reveal a sequentialization scheme of the partial-ordering schedules for TPOL_{SBS} formulas such that all the precedence relations can be verified with a finite amount of information.

Sections 6.2 and 6.3 define the nodes and edges in the tableau, respectively, according to the sequentialization scheme. Section 6.4 wraps everything up to establish the complexity of our algorithm for the TPOL_{SBS} satisfiability problem.

6.1. Sequentialization of partial-ordering schedules

The following structure defines the basic sequentialization scheme of the partial-ordering schedules for TPOL_{SBS} formulas.

Definition 3 (*Knot, full visibility, active visibility*). A *knot* κ in a partial-ordering schedule $\Psi=(\Omega,\Gamma,\Theta,\Lambda)$ for a formula $S\in\text{TPOL}_{\text{SBS}}$ is a set such that $\kappa\subseteq\Omega$, $|\kappa| = H_S$, and for each $1\leqslant i\leqslant H_S$, there is an $a_i\in\kappa$. Given a TPOL formula S and a partial-ordering schedule $\Psi = (\Omega,\Gamma,\Theta,\Lambda)$, a transaction $[p]$ is said to be *fully visible* to a knot κ iff for all $e\in E_{S_*}$, $\Lambda(e,p)\neq\perp$ implies $\exists a_i\in\kappa\exists c\leqslant B_S(\Lambda(e,p) = a_i + c)$. $[p]$ is said to be *actively visible* to κ iff

- for all $e\in E_{S_*}$, $\Lambda(e,p)\neq\perp$ implies $\exists a_i\in\kappa\exists 0\leqslant c\leqslant B_S(\Lambda(e,p) = a_i + c)$; and
- there is an $e\in E_{S_*}$, $\Lambda(e,p)\in\kappa$.

Active visibility is desirable, as will become clear later, in that it makes it possible to evaluate the precedence relations with only a finite amount of information recorded.

We adopt the following notations :

$$\text{next}(\Omega,a_i) = b_i \quad \text{with } b = \min\{b' \mid b'_i\in\Omega; b' > a\}$$

$$\text{last}(\Omega,a_i) = b_i \quad \text{with } b = \max\{b' \mid b'_i\in\Omega; b'\leqslant a\}$$

Given a partial-ordering schedule $\Psi = (\Omega,\Gamma,\Theta,\Lambda)$, we use the procedure $\bar{K}()$ in Table 1 to generate a sequence $\bar{K}(\Psi) = \hat{\kappa}_0\hat{\kappa}_1\ldots\hat{\kappa}_k\ldots$ with the following two nice properties: (1) for every $[p]$, there is a knot in $\bar{K}(\Psi)$ to which $[p]$ is actively visible; and (2) the ordering of quantification is observed in our TPOL semantics. These two properties will be shown in the following two lemma.

Lemma 4. *Given a partial-ordering schedule* $\Psi = (\Omega,\Gamma,\Theta,\Lambda)$, *which satisfies a* TPOL_{SBS} *formula S, for each* $p\in\Theta$, *there is a knot* κ_k *in the sequence generated by* $\bar{K}()$ *in Table* 1 *such that* $[p]$ *is actively visible to* κ_k.

Table 1
Procedure for knot sequence generation

```
K̄(Ψ) /* Ψ = (Ω, Γ, Θ, Λ)*/ {
  (1) k := 0; κ0 := {0_1, ..., 0_{H_S}};
  (2) Loop forever {
        (1) print κ_k;
        (2) Let κ_{k+1} := (κ_k − start(Ψ, κ_k)) ∪ {next(Ω, a_i) | a_i ∈ start(Ψ, κ_k)}
        (3) Let k := k + 1;
      }
}
```

Proof. Due to the progressiveness of partial-ordering schedules, we find that in the sequence $\bar{K}(\Psi)$, there is a least k such that for some $e \in E_S$, $\Lambda(e,p) \in \text{start}(\Psi, \kappa_k)$. We now want to prove that $[p]$ is fully visible to κ_k.

Suppose $[p]$ is not fully visible to κ_k. This means that there are $b_j \in \kappa_k$, $f \in E_{S_*}$, $q \in \Theta$ such that $\Lambda(f,q) = b_j$ and for some $g \in E_{S_*}$, $c > B_S$, $g[p] = b_j + c$. Then according to the first conjunct of the SBS restriction, we know that $\Lambda \models f[q] \preccurlyeq e[p]$. This contradicts the assumption that $\Lambda(e,p) \in \text{start}(\Psi, \kappa_k)$. Thus, we find that $[p]$ must be fully visible to κ_k. Since k is the least such k, we find that the lemma is proven. □

Lemma 5. *Given a partial-ordering schedule* $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$, *which satisfies a* TPOL_{SBS} *formula* S, *for each* $p,q \in \Theta$, *if* $\Gamma \models v \preccurlyeq v'$ *for each* $v \in \text{start}(\Psi, \Lambda[p])$ *and each* $v' \in \text{start}(\Psi, \Lambda[q])$, *then there are* $k \leqslant h$ *such that* $\kappa_k, \kappa_h \in \bar{K}(\Psi)$ *and* $[p], [q]$ *are actively visible to* κ_k *and* κ_h, *respectively.*

Proof. According to Lemma 4, we know the existence of κ_k, κ_h. Since, in statement (2.2) of Table 1, elements are inherited from κ_k by κ_{k+1} according to their precedence, the assumption that

$$\text{for each } p,q \in \Theta, \text{ if } \Gamma \models v \preccurlyeq v' \text{ for each } v \in \text{start}(\Psi, \Lambda[p])$$
$$\text{and each } v' \in \text{start}(\Psi, \Lambda[q])$$

implies that $k \leqslant h$. □

Moreover, the following lemma shows that for each knot in $\bar{K}(\Psi)$, only a finite amount of precedence information is needed to evaluate the temporal precedence relations involved in all the transactions actively visible to the knot.

From now on, given a TPOL formula S, we assume that C_S is the biggest constant used in S.

Lemma 6. *Given a partial-ordering schedule* $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$, *which satisfies a* TPOL_{SBS} *formula* S, *for each* $k \geqslant 0$ *and* $a_i, a'_j \in \kappa_k$, *when* $last(\Omega, a_i - B_S C_S - B_S) \neq \perp$ *and* $next(\Omega, a'_j - C_S) \neq \perp$, $\Gamma \models last(\Omega, a_i - B_S C_S - B_S) \preccurlyeq next(\Omega, a'_j - C_S)$.

Proof. We first want to establish that there are $0 \leqslant c \leqslant d^{\langle i \rangle}, 0 \leqslant c' \leqslant d^{\langle j \rangle}$ ($d^{\langle i \rangle}, d^{\langle j \rangle}$ are defined in the SBS restriction) such that $\Gamma \models a_i - c \asymp a'_j - c'$. According to the second conjunct of SBS restriction, there are $\hat{c} \geqslant 0, \hat{c}' \geqslant 0$ such that $\Gamma \models a_i - \hat{c} \asymp a'_j - \hat{c}'$. We let c be smallest such $\hat{c}$'s and let c' be the smallest such $\hat{c}'$. Now either $c \leqslant d^{\langle i \rangle}$ or $c' \leqslant d^{\langle j \rangle}$ because, otherwise, according to the third conjunct of the SBS restrictions, they cannot be the smallest.

Furthermore, we can show that both $c \leqslant d^{\langle i \rangle}$ and $c' \leqslant d^{\langle j \rangle}$. Assuming that $c' > d^{\langle j \rangle}$, according to the third conjunct of SBS restriction, there are b_i, b'_j such that $\Gamma \models a_i \preccurlyeq b_i \asymp b'_j \preccurlyeq a'_j$. Since $a'_j \in \kappa_k$, b'_j must be in κ_h for some $h < k$. This contradicts the

precedence relation $\Gamma \models a_i \preccurlyeq b_i \asymp b'_j$. Thus, we claim that $0 \leqslant c \leqslant d^{\langle i \rangle}, 0 \leqslant c' \leqslant d^{\langle j \rangle}$, and $\Gamma \models a_i - c \asymp a'_j - c'$.

Now from $a_i - c$ and $a'_j - c'$, we know that there are $1 \leqslant \dot{c} \leqslant d^{\langle i \rangle}$ and $1 \leqslant \dot{c}' \leqslant d^{\langle j \rangle}$ such that $\Gamma \models a_i - c - \dot{c} \asymp a'_j - c' - \dot{c}'$. The minimum value of $\dot{c}'$ is 1 while the maximum value of $\dot{c}$ is B_S according to the second and third conjuncts of the SBS restriction. By repeating this process at most C_S times, we find that $\Gamma \models \text{last}(\Omega, a_i - c - B_S C_S) \preccurlyeq \text{next}(\Omega, a'_j - c' - C_S)$. By choosing the maximum value of c and the minimum value of c', we find that $\Gamma \models \text{last}(\Omega, a_i - B_S C_S - B_S) \preccurlyeq \text{next}(\Omega, a'_j - C_S)$, and the lemma is proven. □

With Lemma 6, we find that we can verify all the temporal precedence relations involved with transactions actively visible to a knot κ iff we can remember the truth value of $a_i + c \preccurlyeq b_j + d$ for each $a_i, b_j \in \kappa, -B_S C_S - C_S - 1 \leqslant c, d \leqslant B_S + C_S + 1$. In the following, we shall propose a tableau-based decision procedure for the satisfiability problem of TPOL_{SBS} formulas. In the tableau graph, we shall construct, say (V, E) where

- each element in V represents a set of knots in partial-ordering schedules for S;
- each element in E represents transitions between different knots as in V.

Each element in V is a pair (P, Q) such that

- P is a finite subset describing the set of transactions actively visible to the knots of this nodes; and
- Q is a consistent set of TPOL formulas with dynamic variables (defined later) which describes the property of this knot set.

Each element in E is a pair (X, Y) such that $X, Y \in V$.

6.2. *Node construction in the tableau*

From now on, we shall suppose that we are given a TPOL_{SBS} formula

$$
S_* \wedge \left(\begin{array}{l}
\bigwedge\limits_{e,f \in E_S} \nabla\, \{[x_1],[x_2]\} \Big(\Big(e[x_1] + d^{\langle \iota(e) \rangle} \preccurlyeq f[x_2] \\
\qquad \rightarrow \bigwedge\limits_{e',f' \in E_S} ((e'[x_1] \neq \perp \wedge f'[x_2] \neq \perp) \rightarrow e'[x_1] \preccurlyeq f'[x_2]) \Big) \Big) \\
\wedge\ \triangle \{[z_1], \ldots, [z_{H_S}]\} \Big(\Big(\bigwedge\limits_{1 < i \leqslant H_S} \pi_1[z_1] \asymp \pi_i[z_i] \Big) \\
\qquad\qquad \wedge \nabla\, \{[x]\} \bigwedge\limits_{e \in E_{S_*}} (e[x] \neq \perp \rightarrow \pi_1[z_1] \preccurlyeq e[x]) \Big) \\
\wedge \bigwedge\limits_{1 \leqslant i,j \leqslant H_S} \nabla\, \{[x_1],[x_2]\} \Big(\pi_i[x_1] \asymp \pi[x_2] \rightarrow \triangle \{[y_1],[y_2]\} \\
\quad \times \Big(\Big(\begin{array}{l} \pi_i[x_1] + d^{\langle i \rangle} \succ \pi_i[y_1] \\ \wedge\ \pi_j[x_2] + d^{\langle j \rangle} \succ \pi_j[y_2] \end{array} \Big) \wedge \pi_i[y_1] \asymp \pi_j[y_2] \Big) \Big)
\end{array} \right)
$$

Our tableau construction should obey the following strategy:

- Tableau-based techniques are applied to S_* and its subformulas.
- Restrictions imposed by the three SBS conjuncts should be enforced by precedence-checking on the transactions actively visible to each knot.

Each node in the tableau represents a set of knots augmented with a set of Boolean variables. We need the following notation to define those augmentative Boolean variables. We first borrow the concept of *dynamic variables* from [1, 22, 23] and define the *dynamic reading variable* T_i+c for each $1 \leqslant i \leqslant H_S$ and $-B_S C_S - C_S - 1 \leqslant c \leqslant B_S + C_S + 1$. Given a $\mathrm{TPOL}_{\mathrm{SBS}}$ formula S, we define the *dynamic reading set* and *ground atom set* of S as

$$\begin{aligned}\mathrm{DTerms}(S) &= \{T_i + c \mid 1 \leqslant i \leqslant H_S; -B_S C_S - C_S - 1 \leqslant c \leqslant B_S + C_S + 1\} \\ &\quad \cup \{T_i^{\ll} \mid 1 \leqslant i \leqslant H_S\} \\ \mathrm{DAtoms}(S) &= \{A \sim B \mid A, B \in \mathrm{DTerms}(S) \cup \{\bot\}; \sim \in \{\preccurlyeq, \prec, \succcurlyeq, \succ, \asymp\}\}\end{aligned}$$

S_1 is called a *subformula* of a $\mathrm{TPOL}_{\mathrm{SBS}}$ S according to the following conditions:

- S_1 is a subformula of $\neg S_1$ and $\triangle\{\ldots\}S_1$.
- Both S_1 and S_2 are subformulas of $S_1 \vee S_2$.
- If S_1 is a subformula S_2 and S_2 is, in turn, a subformula of S_3, then S_1 is also a subformula of S_3.

S_1 is an *extended subformula* of TPOL formula S iff S_1 is identical to a subformula of S except that some of its atoms are replaced by $I \preccurlyeq J$, where I, J are elements in $\mathrm{DTerms}(S)$, or something like $e[x] + c$ with $e \in E_{S_*}, |c| \leqslant C_S$. Now we are ready to define $\mathrm{Closure}(S)$, the finite set of binary variables used to partition the set of knots for S. Given a $\mathrm{TPOL}_{\mathrm{SBS}}$ formula S, $\mathrm{Closure}(S)$ is the smallest set that satisfies the following constraints:

- For all $\phi, \psi \subseteq \mathrm{DTerms}(S)$ and extended subformula S_1 of S_*, $(\phi S_1) \in \mathrm{Closure}(S)$ and $(\psi \phi S_1) \in \mathrm{Closure}(S)$.
- $\mathrm{DAtoms}(S) \subseteq \mathrm{Closure}(S)$.

Given $S_1 \in \mathrm{TPOL}$ and $Q \subseteq \mathrm{Closure}(S)$, S_1/Q is the formula obtained, according to the precedence relation contained in Q, from S_1 by translating each inequality into either *true* or *false* whenever possible. Formally speaking, it is defined in the following way:

$$\begin{aligned}&\mathit{true}/Q = \mathit{true} \\ &(e[x] + c \preccurlyeq f[y] + d)/Q = e[x] + c \preccurlyeq f[y] + d \\ &(T_i + c \preccurlyeq f[y] + d)/Q = T_i + c \preccurlyeq f[y] + d \\ &(T_i^{\ll} \preccurlyeq f[y] + d)/Q = f[y] + d \neq \bot \\ &(e[x] + c \preccurlyeq T_j + d)/Q = e[x] + c \preccurlyeq T_j + d \\ &(f[y] + d \preccurlyeq T_i^{\ll})/Q = \mathit{false} \\ &(T_i + c \preccurlyeq T_j + d)/Q = \begin{cases} \mathit{true} & \text{iff } T_i + c \preccurlyeq T_j + d \in Q \\ \mathit{false} & \text{iff } T_i + c \preccurlyeq T_j + d \notin Q \end{cases}\end{aligned}$$

$$(T_i^{\ll} \leqslant T_i + d)/Q = true$$

$$(T_i + c \leqslant T_i^{\ll})/Q = false$$

$$(\neg S_1)/Q = \neg(S_1/Q)$$

$$(S_1 \vee S_2)/Q = (S_1/Q) \vee (S_2/Q)$$

$$(\triangle\{\ldots\}S_1)/Q = \triangle\{\ldots\}(S_1/Q)$$

With the above notations, we are now ready to establish the correctness of operator $(\ )/Q$. To introduce Lemma 7 and later Lemma 8, we have to extend the environment concept used in defining the semantics of TPOL. Given an environment $\mathscr{E}$ and a knot κ, we shall let $\mathscr{E}^{\kappa}$ be the environment which agrees with $\mathscr{E}$ in every aspect except that for every $1 \leqslant i \leqslant H_S$ and $-B_S C_S - B_S \leqslant c \leqslant B_S + C_S$, when $a_i \in \kappa$, $T_i^{\ll}$ is interpreted as $\text{last}(\Omega, a_i - B_S C_S - B_S)$ and $T_i + c$ is interpreted as $a_i + c$.

Also, given $Q \subseteq \text{Closure}(S)$ and a knot κ in a partial-ordering schedule Ψ, we say Q is *descriptive* of κ in $\tilde{K}(\Psi)$ if for every $a_i, b_j \in \kappa$ and $-B_S C_S - C_S - 1 \leqslant c, d \leqslant B_S + C_S + 1$, $T_i + c \leqslant T_j + d \Leftrightarrow a_i + c \leqslant b_j + d$. Intuitively, Q is *descriptive* of κ in Ψ iff Q symbolically records the precedence relations among the significant events with respect to knot κ.

Lemma 7. *Suppose we are given a* TPOL_{SBS} *formula* S, *a knot* κ *in a partial-ordering schedule* $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ *for* S, *and the set* $Q \subseteq \text{DAtoms}(S)$ *descriptive of* κ *in* Ψ. *Then for each extended subformula* S_1 *of* S, *finite* $\omega \subseteq \Omega$, *and* $\mathscr{E}$, $\Psi : \omega \models_{\mathscr{E}^{\kappa}} S_1$ *iff* $\Psi : \omega \models_{\mathscr{E}^{\kappa}} S_1/Q$.

Proof. Operator $(\ )/Q$ replaces inequalities with truth values only when we encounter atoms like $T_i^{\ll} \leqslant f[y] + d$, $f[y] + d \leqslant T_i^{\ll}$, $T_i + c \leqslant T_j + d$, $T_i^{\ll} \leqslant T_i + d$, and $T_i + c \leqslant T_i^{\ll}$. Because of the interpretative environment $\mathscr{E}^{\kappa}$, Lemma 6, and the definition of Q, we find that operator $(\ )/Q$ indeed preserves the truth values. Thus the lemma is proven. □

For convenience, given a $Q \subseteq \text{Closure}(S)$ descriptive of a knot κ and $\phi \subseteq \text{DTerms}(S)$, we let

$$\text{DStart}(Q, \phi) = \{A \mid A \in \phi; A \neq \perp; \forall B \in \phi(A \succ B \notin Q)\},$$

namely, the set of events corresponding to $\text{start}(\Psi, \omega)$. Similarly,

$$\text{DEnd}(Q, \phi) = \{A \mid A \in \phi; A \neq \perp; \forall B \in \phi(A \leqslant B \notin Q)\},$$

namely the set of events succeeded by no other events in ϕ according to the precedence relation in Q. Then we define the following set to symbolically denote the set of transactions actively visible to any knot of which Q is descriptive :

$$\Pi_S(Q) = \left\{ \lambda \;\middle|\; \begin{array}{l} \forall e \in E_S(\lambda(e,0) = \perp \vee \exists 1 \leqslant i \leqslant H_S \exists 0 \leqslant d \leqslant B_S(\lambda(e,0) = T_i + d)) \\ \forall e \in E_S \forall -C_S \leqslant c \leqslant C_S(\lambda(e,c) \neq \perp \rightarrow \lambda(e,0) + c = \lambda(e,c)) \\ \exists e \in E_S(\lambda(e,0) \in \text{DStart}(Q, \{T_1, \ldots, T_{H_S}\})) \end{array} \right\}$$

Given a TPOL formula S with free name variable p, $Q \subseteq \text{Closure}(S)$, and $\lambda \in \Pi_S(Q)$, we use $S_1^{[x]\leftarrow\lambda}$ to denote the formula identical to S_1 except that each $e[x]+c$ is replaced by $T_i + d$ in case $\lambda(e,c) = T_i + d$. Given a tableau node (P,Q), P is a subset of $\Pi_S(Q)$, and Q is a subset of Closure(S) which satisfies the following *knot consistency conditions* to describe a nonempty knot set:

- $true \in Q$
- Q is descriptive of a knot.
- $P \subseteq \Pi_S(Q)$ and the transactions recorded in P satisfy the three conjuncts of SBS restriction. This can be mechanically checked by examining Q and each transaction in P.
- If $(\phi \neg S_1) \in Q$, then $(\phi S_1) \notin Q$.
- If $(\phi(S_1 \vee S_2)) \in Q$, then at least one of (ϕS_1) and (ϕS_2) is in Q.
- For every $(\phi \triangle \{[x_1],\ldots,[x_m]\}S_1) \in Q$, $(\emptyset\phi \triangle \{[x_1],\ldots,[x_m]\}S_1) \in Q$
- For every $(\psi\phi \bigtriangledown \{[x_1],\ldots,[x_m]\}S_1) \in Q$ and $\lambda \in P$, if $(A \preccurlyeq B \in Q)$ for each $A \in \phi$ and $B \in \text{DStart}(Q, \{\lambda(e,0) \mid e \in E_S\})$, then for each $1 \leqslant i \leqslant m$, $(\text{DEnd}(Q, \psi \cup \text{DStart}(Q, \{\lambda(e,0) \mid e \in E_S\}))\phi(\bigtriangledown\{[x_{i'}] \mid 1 \leqslant i' \leqslant m; i' \neq i\}S_1^{[x_i]\leftarrow\lambda})/Q) \in Q$
- For every $(\psi\phi \triangle \emptyset S_1) \in Q$, $(\psi S_1) \in Q$
- For every $(\psi\phi \bigtriangledown \emptyset S_1) \in Q$, $(\psi S_1) \in Q$

Given an $S \in \text{TPOL}_{\text{SBS}}$, a node (P,Q) in the tableau graph of S is said to be *consistent* iff it satisfies the knot consistency conditions. According to Lemma 4 and the knot consistency condition, it is clear that the SBS restriction can be enforced by checking the finite information contained in each node. Thus, the way we have presented the SBS restriction should only incur minimum verification overhead.

6.3. Edge connections in the tableau

To connect directed edges between pairs of nodes in the tableau graph, we need to define the *dynamic reading variable transformation* which keeps the correct temporal precedence from knots to knots as we go along a path in the tableau graph. This operator determines the correspondence between two dynamic reading variables in different tableau nodes involved in a knot transition. Suppose we are given an extended subformula S_1 of S. Then $S_1^{Q\mapsto}$ is defined inductively in the following way:

- $true^{Q\mapsto} = true$
- $(A \sim B)^{Q\mapsto} = A^{Q\mapsto} \sim B^{Q\mapsto}$
- $(e[x]+c)^{Q\mapsto} = e[x]+c$
- $(T_i + c)^{Q\mapsto}$

$$= \begin{cases} T_i + c & \text{if } T_i \notin \text{DStart}(Q, \{T_1,\ldots,T_{H_S}\}) \\ T_i + c - d & \text{else if } d = \min\{d' \mid T_i \preccurlyeq T_i + d' \in Q\} \wedge c - d \geqslant -B_S C_S - C_S \\ T_i^{\ll} & \text{otherwise} \end{cases}$$

- $(T_i^{\ll})^{Q\mapsto} = T_i^{\ll}$
- $\phi^{Q\mapsto} = \{A^{Q\mapsto} \mid A \in \phi\}$
- $(\neg S_1)^{Q\mapsto} = \neg S_1^{Q\mapsto}$

- $(S_1 \vee S_2)^{Q\mapsto} = S_1^{Q\mapsto} \vee S_2^{Q\mapsto}$
- $(\triangle\{\ldots\}S_1)^{Q\mapsto} = \triangle\{\ldots\}S_1^{Q\mapsto}$

The soundness of the operator $(\ )^{Q\mapsto}$ is established by Lemma 8 in the following.

Lemma 8. *Suppose we are given a partial-ordering schedule* $\Psi = (\Omega, \Gamma, \Theta, \Lambda)$ *for* $S \in \mathrm{TPOL}_{\mathrm{SBS}}$ *and two successive knots* κ, κ' *in the knot sequence* $\bar{K}(\Psi)$ *in Table* 1. *Let* Q *be a subset of* Closure(S) *descriptive of* κ. *Then for every finite* $\omega \subset \Omega$ *in* Ψ *and* $S_1 \in \mathrm{Closure}(S)$, $\Psi : \omega \models_{\mathscr{E}^\kappa} S_1$ *iff* $\Psi : \omega \models_{\mathscr{E}^{\kappa'}} S_1^{Q\mapsto}$.

Proof. The proof is very much similar to the one for the change of initial time lemma used in [1]. □

We say $(\psi\phi \triangle \{[x_1], \ldots, [x_m]\}S_1) \in Q$ is *fulfilled* at node (P, Q) iff there are $\lambda \in P$ and $1 \leqslant i \leqslant m$ such that

- $A \preccurlyeq B \in Q$ for every $A \in \phi$ and $B \in \mathrm{DStart}(Q, \{\lambda(e, 0) \mid e \in E_S\})$; and
- $(\mathrm{DEnd}(Q, \psi \cup \mathrm{DStart}(Q, \{\lambda(e, 0) \mid e \in E_S\}))\phi(\triangledown\{[x_{i'}] \mid 1 \leqslant i' \leqslant m; i' \neq i\}S_1^{[x_i]\leftarrow\lambda})/Q) \in Q$

Given two nodes (P_1, Q_1) and (P_2, Q_2), we connect an arc from the former to the latter iff the following three conditions are satisfied:

- For every $A \sim B \in Q_1$, $(A \sim B)^{Q_1\mapsto} \in Q_2$.
- For every unfulfilled $(\psi\phi \triangle \{\ldots\}S_1) \in Q_1$, $(\psi^{Q_1\mapsto}\phi^{Q_1\mapsto}((\triangle\{\ldots\}S_1)^{Q_1\mapsto}/Q_2)) \in Q_2$.
- For every $(\psi\phi \triangledown \{\ldots\}S_1) \in Q_1$, $(\psi^{Q_1\mapsto}\phi^{Q_1\mapsto}((\triangledown\{\ldots\}S_1)^{Q_1\mapsto})/Q_2) \in Q_2$.

6.4. *Answering the satisfiability problem*

Based on our definition of tableau node structure and edge connection, we now propose the following condition on the tableau graph for determining the eventual fulfillment of $\triangle$-formulas. Given a $\mathrm{TPOL}_{\mathrm{SBS}}$ formula S and $(\psi_0\phi_0S_0) \in \mathrm{Closure}(S)$, a finite sequence $(\psi_0\phi_0S_0)(\psi_1\phi S_1)\ldots(\psi_m\phi_mS_m)$ is called a *fulfilling sequence* for $(\psi_0\phi_0S_0)$ in the tableau graph for S along a finite sequence $N_0 \to N_1 \to \cdots \to N_m$ in the tableau graph for S with $N_k = (P_k, Q_k)$ iff

- $(\psi_k\phi_kS_k) \in Q_k$, for each $0 \leqslant k \leqslant m$;
- for each $0 \leqslant k < m$, $(\psi_k^{Q_k\mapsto}\phi_k^{Q_k\mapsto}(S_k^{Q_k\mapsto})/Q_{k+1}) = (\psi_{k+1}\phi_{k+1}S_{k+1})$; and
- $(\psi_m\phi_mS_m) \in Q_m$ is fulfilled at N_m.

Given a $\mathrm{TPOL}_{\mathrm{SBS}}$ formula S, an infinite path $\sigma = N_0 \to N_1 \to \cdots \to N_k \to \cdots$, where $N_k = (P_k, Q_k)$, in its tableau is called a *satisfying path* iff

- (*Initialization*) $(\{T_1\}S_*) \in Q_0$;
- (*Fulfillment of eventuality*) for every $k \geqslant 0$ and $(\psi\phi \triangle \{[x_1], \ldots, [x_m]\}S_1) \in Q_k$, there is a fulfilling sequence for $(\psi\phi \triangle \{[x_1], \ldots, [x_m]\}S_1) \in Q_k$ along a finite node sequence along σ starting at N_k.

Finally, we need the following notation to simplify our proof for the correctness of tableau graph construction. Suppose we are given a subformula of S, an environment $\mathscr{E}$, and a knot κ. Formula translation Dynamic($S_1, \mathscr{E}, \kappa$) is identical to S_1 except for each

term $e[x]+c$ in S_1, if x is interpreted in $\mathscr{E}$ and $\Lambda(e,\mathscr{E}(x))+c=a_i+d$ for some $a_i\in\kappa$ and $h\in\mathscr{Z}$, replace the term by $T_i^{\ll}$ in case $d<-B_SC_S-B_S$; and by T_i+d in case $d\geqslant -B_SC_S-B_S$. Also, given a set ϕ of terms like $e[x]$, we let $\text{Dynamic}(\phi,\mathscr{E},\kappa)=\{\text{Dynamic}(A,\mathscr{E},\kappa)\mid A\in\phi\}$.

Lemma 9. *A* TPOL_{SBS} *formula is satisfiable iff its tableau has a satisfying path.*

Proof. ($\Rightarrow$) First we want to show that if S is satisfiable, then its tableau graph contains a satisfying path. Since S is satisfiable, we know that there is a partial-ordering schedule $\Psi=(\Omega,\Gamma,\Theta,\Lambda)$ which satisfies S. We want to show that, corresponding to Ψ, there is a satisfying path $\sigma=N_0\rightarrow N_1\rightarrow\cdots\rightarrow N_k\rightarrow\cdots$ in the tableau graph for S such that, for each membership relation $(\phi S_1)\in Q_k$ necessary for the condition of the satisfying path, a corresponding satisfaction relation $\Psi:\omega\models_{\mathscr{E}} S_1$ is incurred in the satisfaction requirement of S along Ψ. Formally speaking, σ must satisfy the following conditions:

(1) For each $k\geqslant 0$, κ_k is defined in procedure $\bar{K}(\Psi)$ in Table 1.

(2) For each $k\geqslant 0$ and $[p]$ actively visible to κ_k, there is a $\lambda\in P_k$ such that for every $e\in E_S$, $\Lambda(e,p)=a_i+c$ for some $a_i\in\kappa_k$ iff $\lambda(e,0)=T_i+c$.

(3) For each $k\geqslant 0$, Q_k is descriptive of κ_k.

(4) For each $\Psi,\omega\models_{\mathscr{E}} S_1$, there is a $k\geqslant 0$ such that $\forall a_i\in\omega\exists b_i\in\kappa_k\exists c\leqslant B_S(a_i=b_i+c)$, $(\text{DEnd}(Q_k,\text{Dynamic}(\omega,\mathscr{E},\kappa_k))(\text{Dynamic}(S_1,\mathscr{E},\kappa_k))/Q_k)\in Q_k$.

To show the existence of σ, we need to show the existence of both nodes and arcs in the tableau graph. The existence of nodes in σ in the tableau graph can be easily checked because the above-mentioned four conditions are all satisfiable under the definition of our tableau nodes. Especially, condition (2) is satisfiable according to Lemma 4. Condition (3) is satisfiable because of the consistency of the partial-ordering relation in Γ. Condition (4) is satisfiable because, for all $k\geqslant 0$, $Q_k\subseteq\text{Closure}(S)$.

As for the existence of the arcs connecting the nodes in σ, this can be satisfied because of the inductive semantics of TPOL, because of the translation of formulas along the edges, and because of Lemmas 7 and 8.

Finally, σ is indeed a satisfying path because it is derived from Ψ, and all eventualities in Ψ will be satisfied, and the elements in Q_k, for all k, are there because the corresponding extended subformulas are satisfied by Ψ.

($\Leftarrow$) Secondly, we want to show that, if the tableau graph of a well-formed TPOL_{SBS} formula S contains a satisfying path, then S is satisfiable. Assume that there is a satisfying path $\sigma=N_0\rightarrow N_1\rightarrow\cdots\rightarrow N_k\rightarrow\cdots$ in the tableau graph for S. Now from σ, we shall construct a partial-ordering schedule $\Psi=(\Omega,\Gamma,\Theta,\Lambda)$ which satisfies S. Formally speaking, Ψ must satisfy the following conditions:

(A) The precedence relations Γ must satisfy the condition that, for each $k\geqslant 0$ and $a_i\in\kappa_k$ in $\bar{K}(\Psi)$ in Table 1, $a_i\in\kappa_{k+1}$ iff $T_i\notin\text{DStart}(Q_k,\{T_1,\ldots,T_{H_S}\})$.

(B) For each $k\geqslant 0$ and $\lambda\in P_k$, there is a $[p]$ actively visible to κ_k such that for every $e\in E_S$, $\Lambda(e,p)=a_i+c$ for some $a_i\in\kappa_k$ iff $\lambda(e,0)=T_i+c$.

(C) For each $k\geqslant 0$, Q_k is descriptive of κ_k in Ψ.

(D) For each $k \geqslant 0$, subformula S_1 of S, environment $\mathscr{E}$, and finite $\omega \subseteq \Omega$ such that $\forall a_i \subset \omega \exists b_i \in \kappa_k \exists c \leqslant B_S(a_i = b_i + c)$,

$$\Psi, \omega \models_{\mathscr{E}} S_1 \text{ iff } (\mathrm{DEnd}(Q_k, \mathrm{Dynamic}(\omega, \mathscr{E}, \kappa_k))(\mathrm{Dynamic}(S_1, \mathscr{E}, \kappa_k))/Q_k) \in Q_k.$$

To show the existence of Ψ, we notice that, in condition (A), the κ_k's monotonically advance their reading values. In condition (B), a transaction $[p]$ parallel to λ is definable because of Lemma 4. In condition (C), the temporal precedence relation Γ is definable since, for each node, say (P, Q), in the tableau graph, $Q \cap \mathrm{DAtoms}(S)$ is consistent. Furthermore, the arcs among the nodes incrementally add and delete a consistent precedence relation to the nodes, and Lemma 8 shows that the local clock reading translation among the nodes preserves the temporal precedence. In condition (D), the requirements from different nodes on the same subformula are consistent because of Lemma 8. Also, the satisfaction of each eventuality is also guaranteed because of the fulfilling sequences embedded in the satisfying path for each $\triangle$-formula and because of the second bullet of edge connection.

Thus the lemma is proven. □

Finally, the following theorem states the complexity of the $\mathrm{TPOL}_{\mathrm{SBS}}$ satisfiability problem.

Theorem 10. *The* $\mathrm{TPOL}_{\mathrm{SBS}}$ *satisfiability problem is* EXPSPACE-*complete.*

Proof. The "hardness" part of the proof is a straightforward copy of the proof of Theorem 1 and follows from Corollary 3. The "easiness" part of the proof follows the same pattern of the completeness proof in [1, 22] because we can show that the size of Closure(S) is $O(2^{|S|})$ and any node consistency check, and the edge connection check can be done linear to the size of tableau graph. □

7. Conclusion

The concept of transactions is widely accepted in specification of computer systems. TPOL allows quantification on transactions and precise description of interaction and internal structures of transactions. In summary, it has the following potential advantage for designing specification languages for distributed real-time systems with special syntax and high-level semantics:

- *Necessary expressiveness for partial-ordering in distributed real-time systems.*
- *A solid theory for specification and verification for modern computer systems.*
- *Better representation of higher-level human reasoning in system design.*
- *More built-in semantics and smaller verification overhead.*

As an old Chinese metaphor goes, expressiveness and verifiability are like the two edges of a sword in language design. Moving too much in one direction can hurt the practicality of the language design. As computer technology continues to advance, the

appropriate balance between expressiveness and verifiability of specification languages will also continue to change.

References

[1] R. Alur and T.A. Henzinger, A really temporal logic, in: *Proc. 30th IEEE Symp. Found. of Computer Sciences* (1989) 164–169.

[2] M. Boyer, *A Computational Logic Handbook* (Academic Press, New York, 1988).

[3] E.M. Clarke and E.A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, in: *Proc. of Workshop on Logics of Programs*, Lecture Notes in Computer Science, Vol. 131 (Springer, Berlin, 1981) 52–71.

[4] E. Clarke, E.A. Emerson and A.P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, *ACM Trans. Programming Languages and Systems* **8** (1986) 244–263.

[5] C. Chang and R.C. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, New York, 1973).

[6] R.W. Floyd, *New Proofs and Old Theorems in Logic and Formal Linguistics* (Computer Associates Inc., Wakefield, MA, 1964).

[7] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[8] F. Jahanian and A.K. Mok, Safety analysis of timing properties in realtime systems, *IEEE Trans. Software Eng.*, **SE-12** (1986) 890–904.

[9] F. Jahanian and A.K. Mok, A graph theoretic approach for timing analysis and its implementation, *ACM Trans. Comput.* **C-36** (1987) 961–975.

[10] F. Jahanian and A.K. Mok, Modechart: a specification language for real-time systems, *IEEE Trans Software Eng.* **20** (1994) 933–947.

[11] C.B. Jones, *Systematic Software Development using VDM* (Prentice-Hall, Englewood Cliffs, NJ, 2nd eds., 1990).

[12] C.B. Jones, A pi-calculus semantics for an object-based design notation, CONCUR, 1993.

[13] L. Lamport, Time, clocks, and the ordering of events in a distributed systems, *Comm. ACM* 21 (1978) 558–565.

[14] L. Lamport, Sometimes is sometimes "Not never"-on the temporal logic of programs, in: *Proc. 7th Ann. ACM Symp. on Principles of Programming Languages* (1980) 174–185.

[15] A. Pnueli, The temporal logic of programs, in: *Proc. 18th Ann. IEEE-CS Symp. on Foundations of Computer Science* (1977) 45–57.

[16] E. Post, A variant of a recursively unsolvable problem, *Bull. AMS* **52**, 264–268.

[17] R.S. Pressman, *Software Engineering, A Practitioner's Approach* (McGraw-Hill, New York, 1982).

[18] J.A. Robinson, A machine oriented logics based on the resolution principles, *J. ACM* **12** (1965) 23–41.

[19] A. Silberschatz and J.L. Peterson, *Operating System Concepts* (Alternate Edition) (Addison-Wesley, Reading, MA, 1988).

[20] J.M. Spivey, *The Z Notation, A Reference Manual* (Prentice Hall, Englewood, Cliffs, NJ, 2nd edn., 1992).

[21] J.A. Stankovic, A serious problem for next-generation systems, *IEEE Comput.* **21** (1988) 10–19.

[22] F. Wang and A.K. Mok, Asynchronous real-time event logic, in: *Proc. Internat. Computer Symposium*, Taiwan, Vol. 1 (1992) 77–84.

[23] F. Wang and A.K. Mok, RTL and refutation by positive cycles, in: *Proc. Formal Method Europe Symp.*, Spain, 1994, ao appear.

[24] F. Wang, A. Mok and E.A. Emerson, Distributed real-time system specification and verification in APTL, *ACM Trans. Software Engineering and Methodology*, 1993.